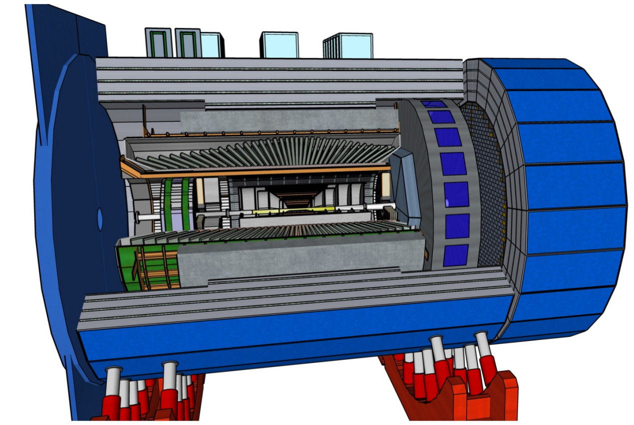
ECCE



The DAQ and Online System for the ~~ECCE~~ EPIC Proposal at EIC

Martin L. Purschke

BROOKHAVEN NATIONAL LABORATORY

After the ECCE proposal was chosen to be "Detector 1", just last week we voted to call the experiment "EPIC" - s/ECCE/EPIC/g

Manhattan

Long Island, NY

RHIC/EIC from space

The ECCE DAQ

Conveners during the ECCE propsal: Chris Cuevas (Jlab) and Martin Purschke (BNL)

Chris continues, and Jo Schambach (ORNL) has taken over from me (I got fired ☺)

(My real day job is the sPHENIX DAQ manager)

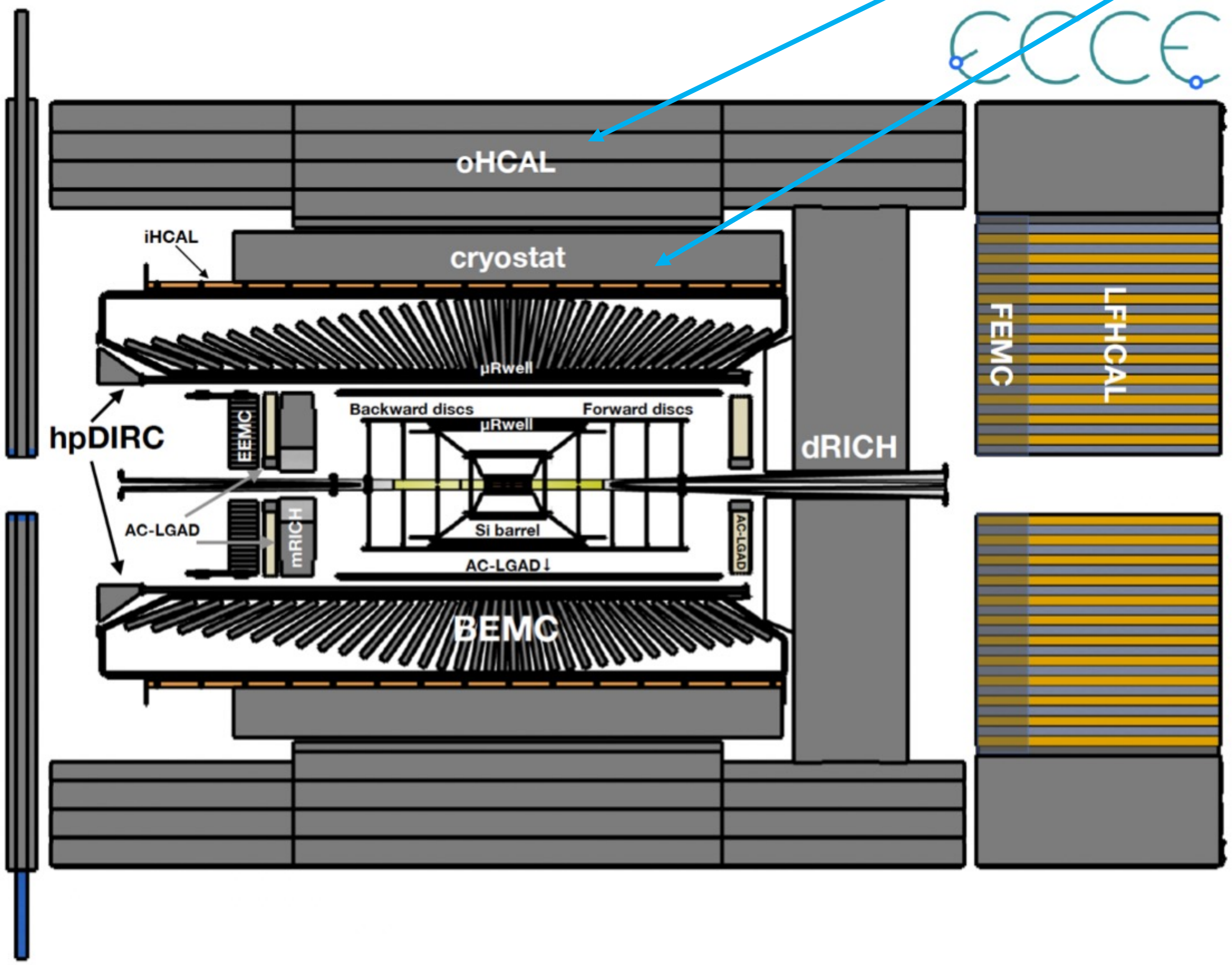
The ECCE DAQ and Timing system is heavily influenced by the corresponding sPHENIX systems

Some personnel overlap, but also

- Key concepts (Streaming Readout, the use of ASICs, use of a “DAM” (FELIX in 2022)) are very similar. DAM = “Data Aggregation Module"
- Low-jitter clock distribution like sPHENIX’s to a FELIX successor is a key ingredient
- Concept is designed with a distributed calibration/reconstruction paradigm (Grid) in mind
- It’s scalable, and has well-defined hand-off points where common technologies (transports, storage, monitoring and other APIs) take over

From the Proposal

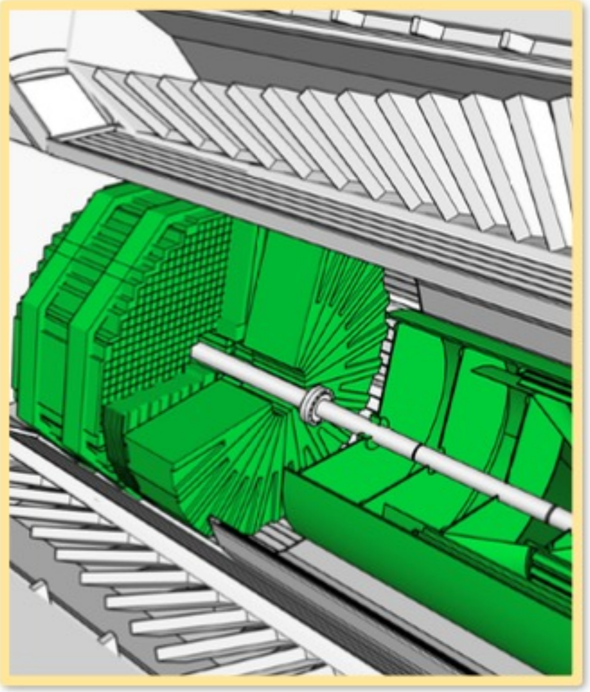
The current sPHENIX carriage with the outer HCal and the magnet

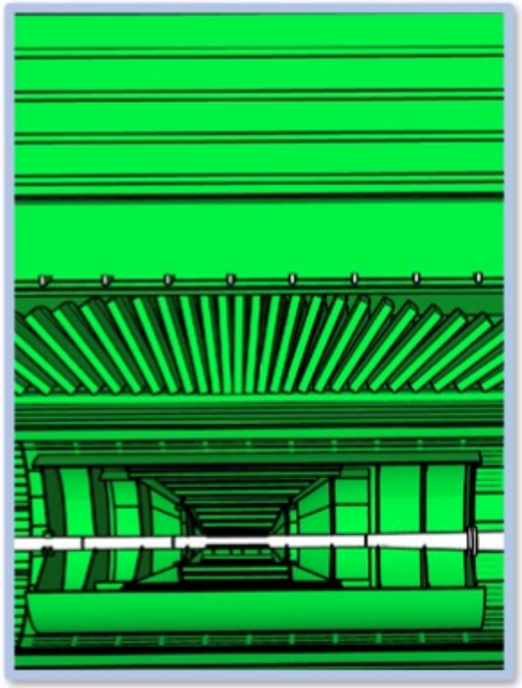


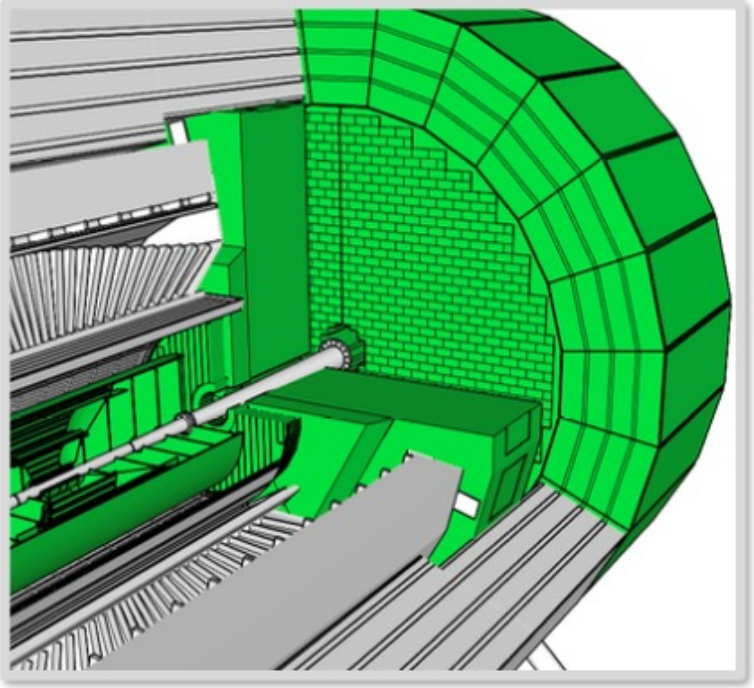
Hadron Beam

Electron Beam

From the Proposal







Backward Endcap

Tracking:
- ITS3 MAPS Si discs (x4)
- AC-LGAD

PID:
- mRICH
- AC-LGAD TOF
- $PbWO_4$ EM Calorimeter (EEMC)

Barrel

Tracking:
- ITS3 MAPS Si (vertex x3; sagitta x2)
- μRWell outer layer (x2)
- AC-LGAD (before hpDIRC)
- μRWell (after hpDIRC)

h-PID:
- AC-LGAD TOF
- hpDIRC

Electron ID:
- SciGlass EM Cal (BEMC)

Hadron calorimetry:
- Outer Fe/Sc Calorimeter (oHCAL)
- Instrumented frame (iHCAL)

Forward Endcap

Tracking:
- ITS3 MAPS Si discs (x5)
- AC-LGAD

PID:
- dRICH
- AC-LGAD TOF

Calorimetry:
- Pb/ScFi shashlik (FEMC)
- Longitudinally separated hadronic calorimeter (LHFCAL)

Rough Subsystem Count

- ~ 20 different detector components combined in the 3 parts (backward/forward/central barrel)
- Just a quick overview for reference here (backward/forward), read all about it in the proposal

Topic	Challenge	ECCE solution	Comment
Far-Backward – Low-Q^2 Tagger	Measure low-Q^2 photo-production with as minimal a Q^2-gap as possible.	Spectrometer with AC-LGAD tracking and $PbWO_4$ calorimetry	
Far-Backward – Luminosity Detector	e-ion collision luminosity to better than 1% and relative Luminosity for spin asymmetries to 10^{-4}	Zero Degree Calorimeter with x-ray absorber and e^+/e^- pair spectrometer with AC-LGAD tracking and $PbWO_4$ calorimetry	two complementary detection systems
Far-Forward – B0 Spectrometer	$\eta > 4$ charged particle tracking and γ measurement	Four Si trackers with 10 cm $PbWO_4$ calorimeter	
Far-Forward – Off-momentum Detectors	forward particles (Δ, Λ, Σ, etc) decay product measurement	AC-LGAD detectors	Sensors on one side detect p, on other side p^- from Λ decay; sensors outside beam pipe
Far-Forward – Roman Pots	Detect low-p_T forward-going particles	AC-LGAD detectors	fast timing (∼35 ps) removes vertex smearing effects from crab rotation; 10σ from beam
Far-Forward – Zero-degree Calorimeter	Measure forward-going neutrons γ and heavy-ion fission product	FOCAL-type calorimeter with high-precision EM and Hadron Calorimetry	Upgrade option: AC-LGAD layer to capture very high rapidity charged tracks

An idea of the channel count

PID WBS Name	Detector	ASIC	Channels
Barrel PID	hpDIRC	High Density SoC	69,632
	TOF	eRD112 development	8,600,000
Electron Endcap	mRICH	High Density SoC	65,536
	TOF	eRD112 development	920,000
Hadron Endcap	dRICH	MAROC3	19,200
	TOF	eRD112 development	1,840,000
Far-Forward Detectors	Roman Pots	eRD112 development	524,288
	B0 Detector	eRD112 development	2.6M
	Off-Momentum Detectors	eRD112 development	1.8M
Far-Backward Detectors	Low-Q^2 Tagger	eRD112 development	4.6M
	Luminosity Monitor	eRD112 development	268,441

Channels
69,632
8,600,000
65,536
920,000
19,200
1,840,000
524,288
2.6M
1.8M
4.6M
268,441

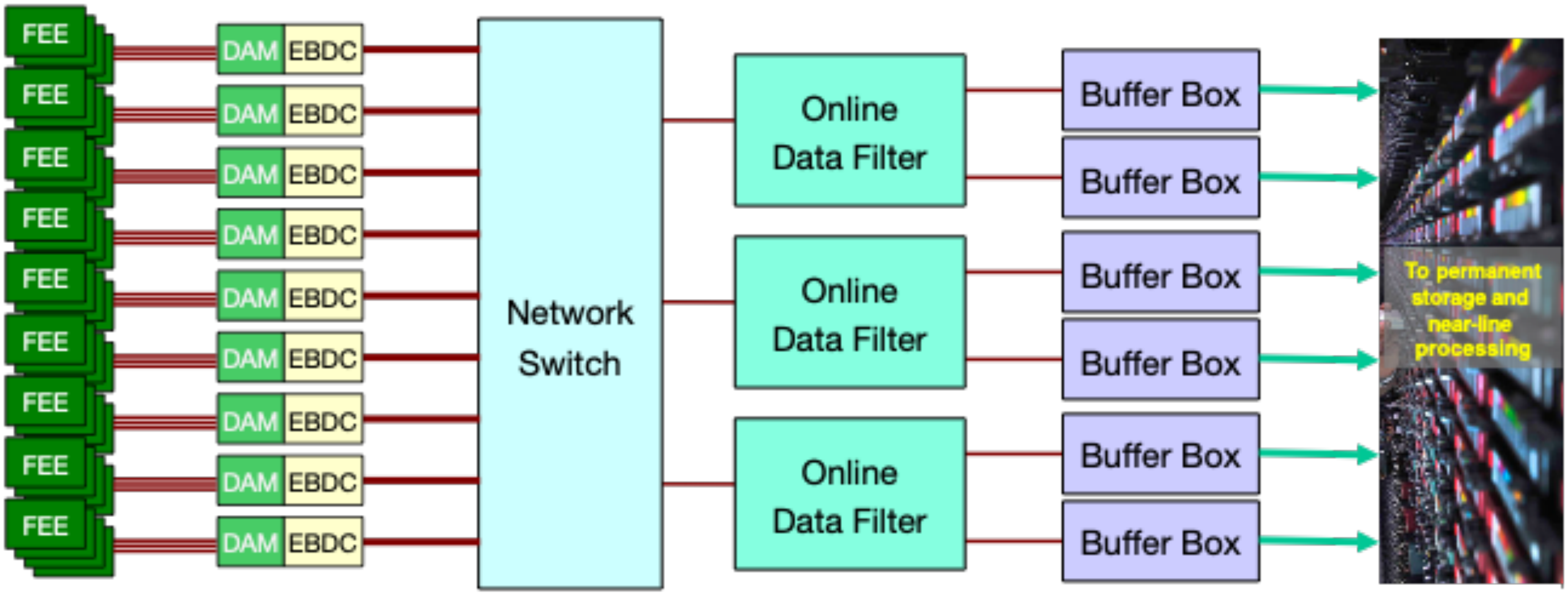
Makes about 21 million channels in round numbers

Data statistics

	ECCE Runs		
	year-1	year-2	year-3
Luminosity	$10^{33}\text{cm}^{-2}\text{s}^{-1}$	$2 \times 10^{33}\text{cm}^{-2}\text{s}^{-1}$	$10^{34}\text{cm}^{-2}\text{s}^{-1}$
Weeks of Running	10	20	30
Operational efficiency	40%	50%	60%
Disk (temporary)	1.2 PB	3.0 PB	18.1 PB
Disk (permanent)	0.4 PB	2.4 PB	20.6 PB
Data Rate to Storage	6.7 Gbps	16.7 Gbps	100 Gbps
Raw Data Storage (no duplicates)	4 PB	20 PB	181 PB
Recon process time/core	5.4 s/ev	5.4 s/ev	5.4 s/ev
Streaming-unpacked event size	33kB	33kB	33kB
Number of events produced	121 billion	605 billion	5,443 billion
Recon Storage	0.4 PB	2 PB	18 PB
CPU-core hours (recon+calib)	191M core-hours	953M core-hours	8,573M core-hours
2020-cores needed to process in 30 weeks	38k	189k	1,701k

To put the red box into context – sPHENIX will write 1.5PB/day, 9PB a week – compare to 6PB/week here, in 2033 or so

DAQ Bird's eye view

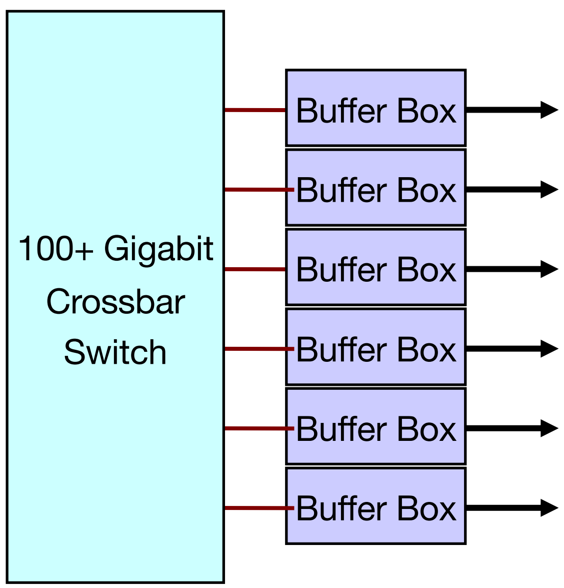


Only a few elements shown

FEEs vary a lot, complexity varies a lot, data volume varies a lot

Common denominator is that there is a uniform data structure at the output of the DAM

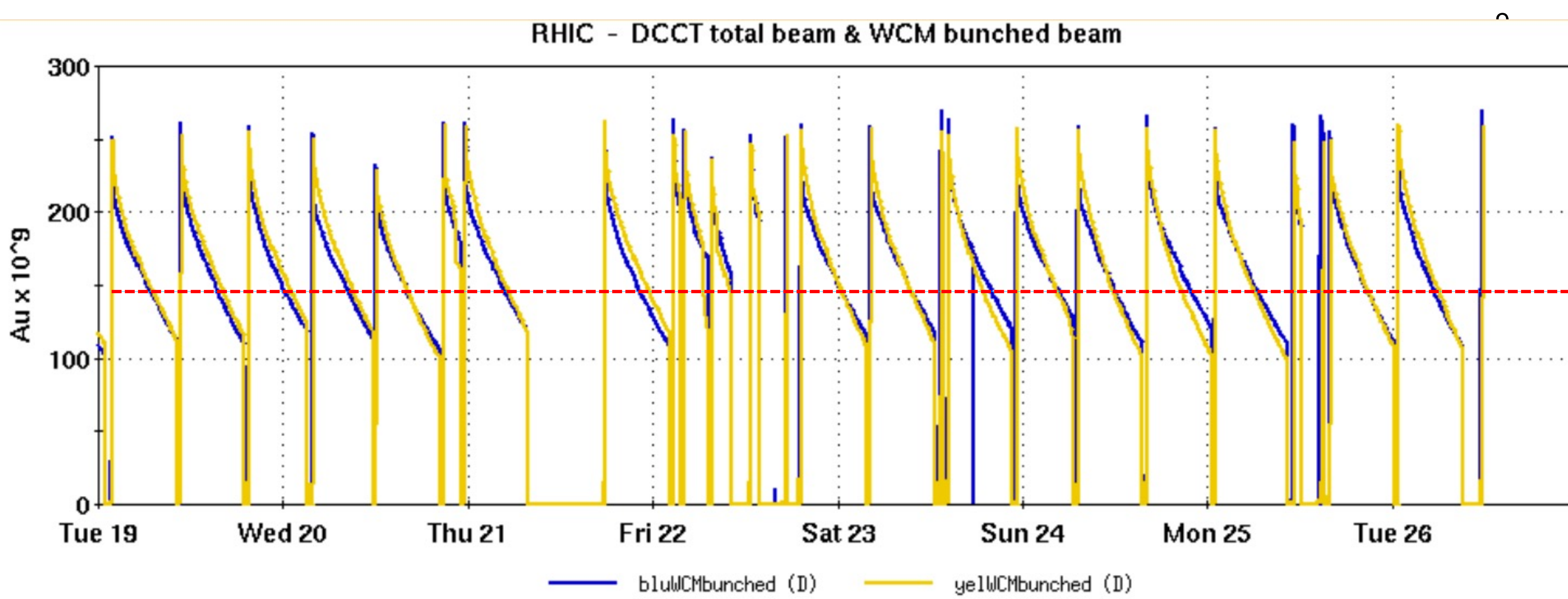
Why do we call those “BufferBoxes”?



The data rate at a collider is “bursty” – high luminosity at the begin of a store, then ”burning off” – change of a factor of 2

Also gaps in data flowing with collider dump/fill, access, APEX, MD

This Buffer boxes allow us to send the average, rather than the peak rate through the WAN



2016 (last PHENIX run) beam intensity over a week

Average

Streaming readout, here we come!

Past the FEE, the readout is completely oblivious to the readout mode

It doesn't care how the front-end arrived at the decision to send up the data.

Triggered or streaming, from the readout perspective they look the same

I have come to regard a particular feature of SRO as the defining property, even if you ultimately trigger your front-end:

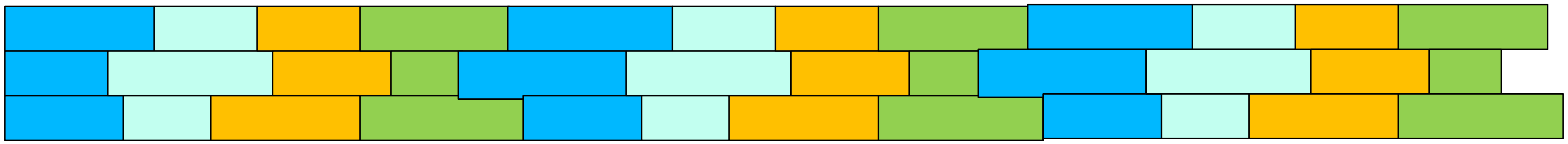
There is no synchronized end to a given event!

While "event" *n* is streaming, in other places, event *n-1* (or *-2, -3, -4…*) isn't finished yet, and data from different crossings are interleaved

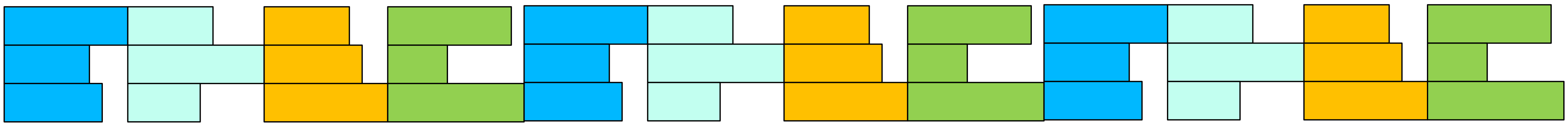
And that's where the speed increase can be significant even for "classic" systems

Offline sorting streaming data into events/crossings

A “streaming data” offline **pool** holds a number of crossings worth of data (like 1000) and sorts them by crossing number (beam clock value)



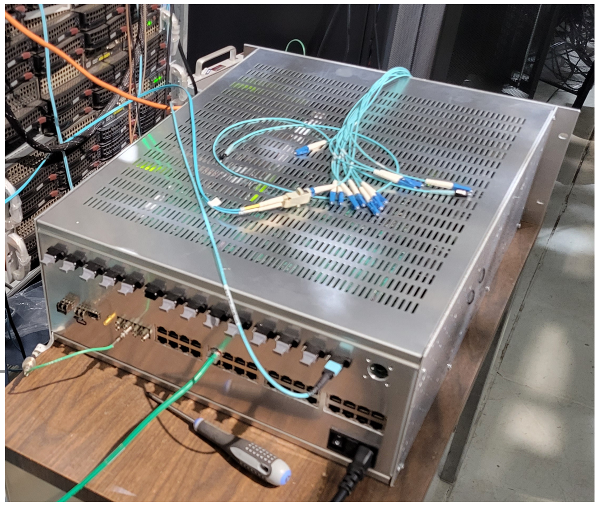
It then hands out per-crossing data:



As processed data (oldest crossings) get discarded, new data are inserted (high-and low-water marks)

But how do you “sort”? With the Timing system!

Timing System



Pick a convenient multiple of the beam clock frequency (sPHENIX: 6)

Have a global, never-reverting master BCO counter – 64 bit, transmit BCO LSBs to front-ends (40 bits)

Front-ends embed a number of those bits in their lower-level data structures (Felix - 40, FEE - 20)

The **only way** to send information to the FEE on a per-crossing basis (like, have the FEE or DAM do something different in the abort gap)

One beam crossing

Bit Number	Function	Beam clock phases					
		0	1	2	3	4	5
7-0	Mode bits /BCO	Modebits bits 7-0	BCO bits 7-0	BCO bits 15-8	BCO bits 23-16	BCO bits 31-24	BCO bits 39-32
8	Beam clock phase0	1	0	0	0	0	0
9	LVL1 accept	X	0	0	0	0	0
10	Endat 0	X	X	X	X	X	X
11	Endat 1	X	X	X	X	X	X
12	Modebit enable	1	0	0	0	0	0
15-13	User bits	3 user bits	0	1	2	3	4

40 bits BCO

Example: sPHENIX TPC data

Clock values embedded in FEE data

40 bits BCO

```
0000000 feee ba5e 0ff1 0001 7229 f7a0 0088 0004
0000020 002f 8782 0004 ffff 0081 0000 0050 0050
…
0001020 d72c 0081 feed 0000 0088 3e2b 0004 feed
0001040 000f 0088 9f7a 0000 0000 0007 ffff 58af
…
0002100 0088 ad79 0004 feed 0017 0088 9f7a 0000
0002120 0000 000f ffff 58af 0081 0008 0000 ffff
…
0004740 0004 feed 0027 0047 0088 9f7a 0000 8782
0004760 0000 0004 001f ffff ffff 58af 0000 0000
```

FELIX Hdr

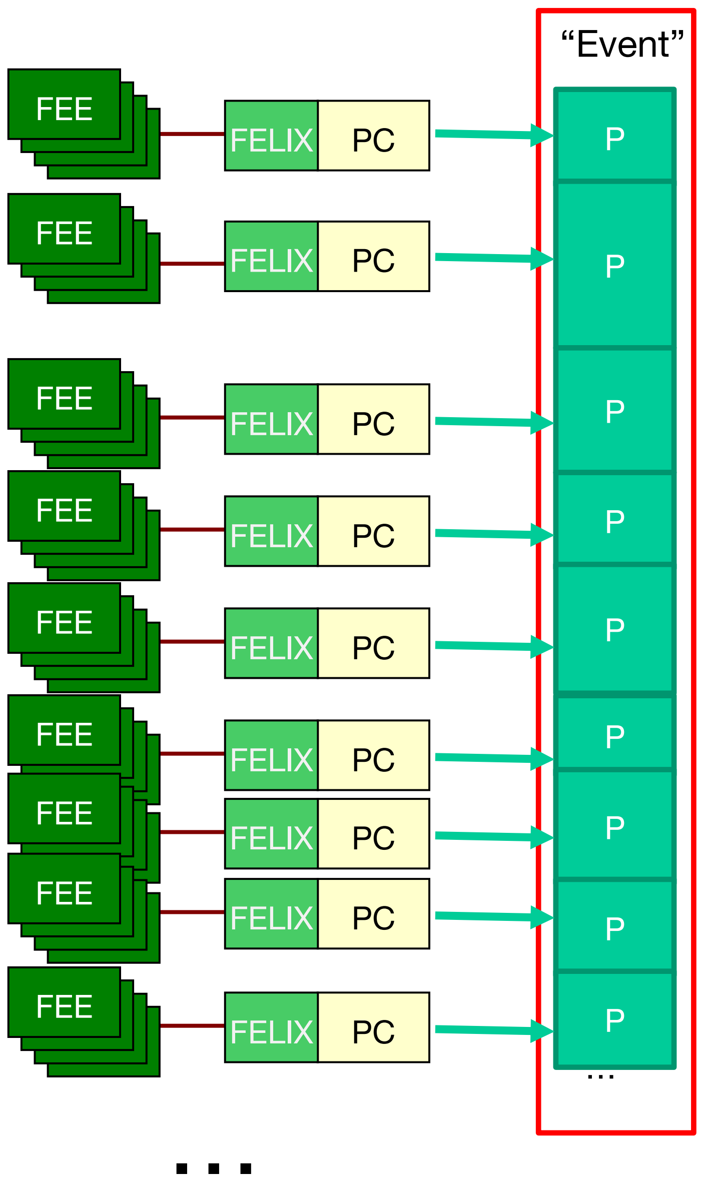
FEE structures

Clock values

```
bx 9f7a0
bx 9f7a0
bx 9f7a0
bx 9f7a0
…
```

In this way you can verify the integrity of the internal data structures, and sort the data by “time”

Event / Streaming Data Structures



Each Front-End Card generally contributes what we call a "Packet" to the overall event structures

A Packet ID uniquely identifies the detector component / front-end card where it comes from

A hitformat field identifies the format of the data, und ultimately selects the decoding algorithm

You interact with a standard set of APIs to access the data

We can change/improve the binary format and assign a new hitformat for a packet at any time

Insulation of offline software from changes in the online system

API delivers the data independent of internal encoding

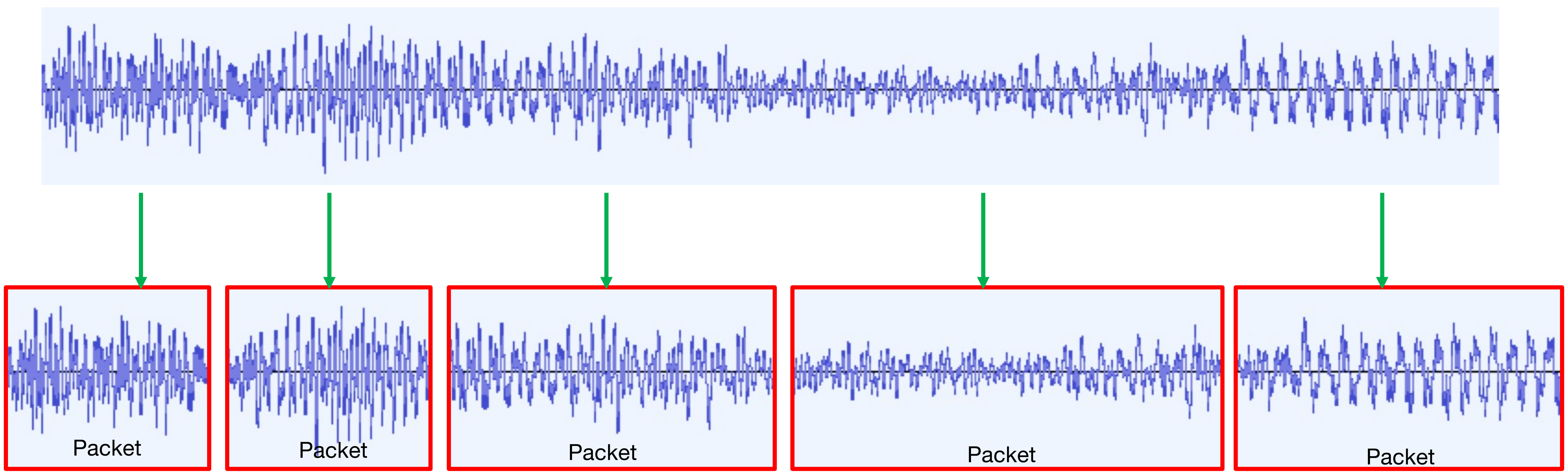
Very rough number: 1200 packets collectively

In case of a triggered DAQ, such an event structure and the packets therein would correspond to the data from one crossing

Streaming Readout and Packets

For streaming data, the “Packet” paradigm changes its meaning a bit

It becomes like a packet in the Voice-Over-IP sense - VoIP is chopping an audio waveform into conveniently-sized chunks to transfer through a network



We are chopping the streaming detector data into conveniently-sized packets for storage

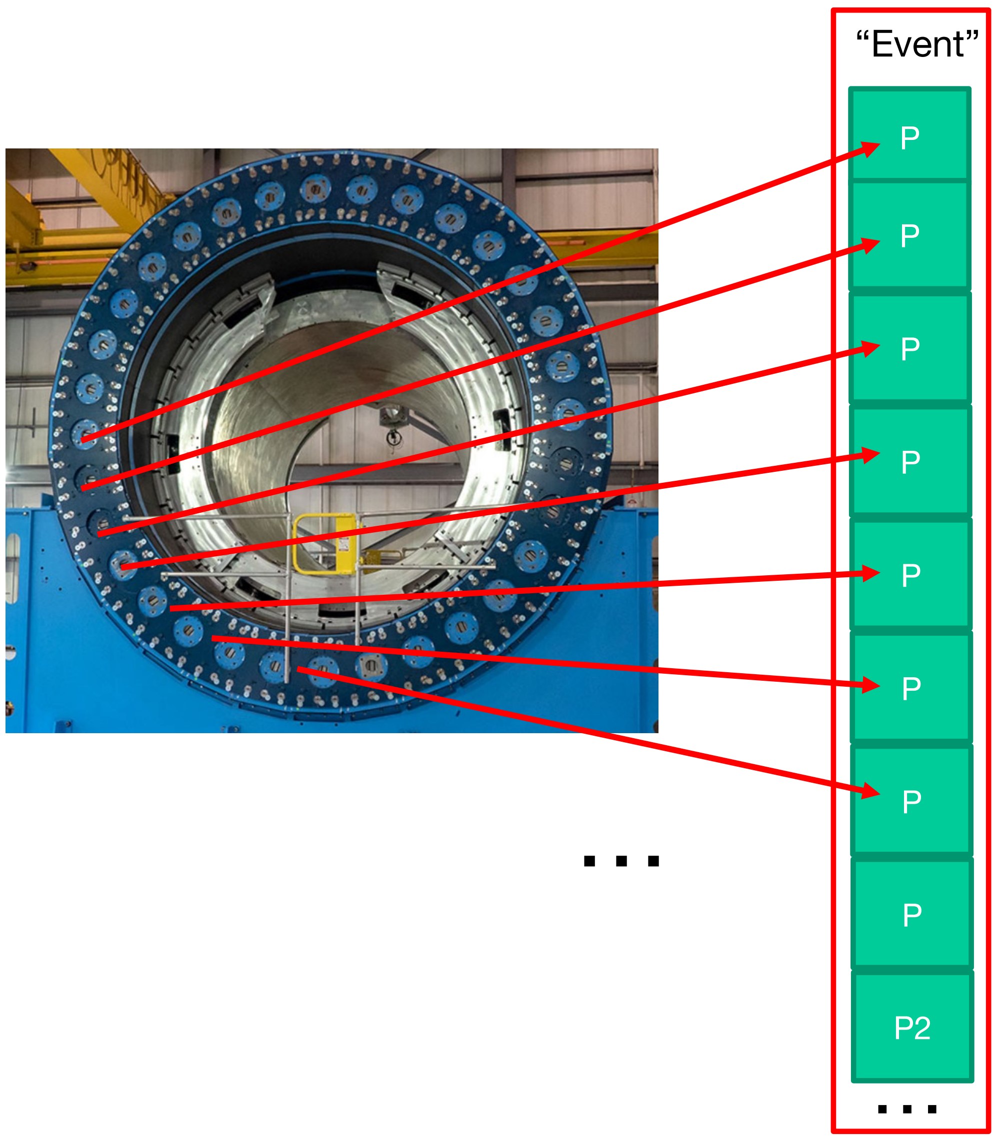
Here: Streaming sPHENIX TPC data (entire sPHENIX tracking system streams!)

```
$ dlist  rcdaq-00002343-0000.evt -i
 -- Event     2 Run:  2343 length: 5242872 type:  2 (Streaming Data)  1550500750
Packet  3001 5242864 -1 (sPHENIX Packet)  99 (IDTPCFEEV2)
$
```

Example: Full EPIC Outer HCal, Real Events

That’s one of the detectors that will survive into Detector 1

For us it’s subsystem #8, makes 32 Packets with IDs 8001 - 8032



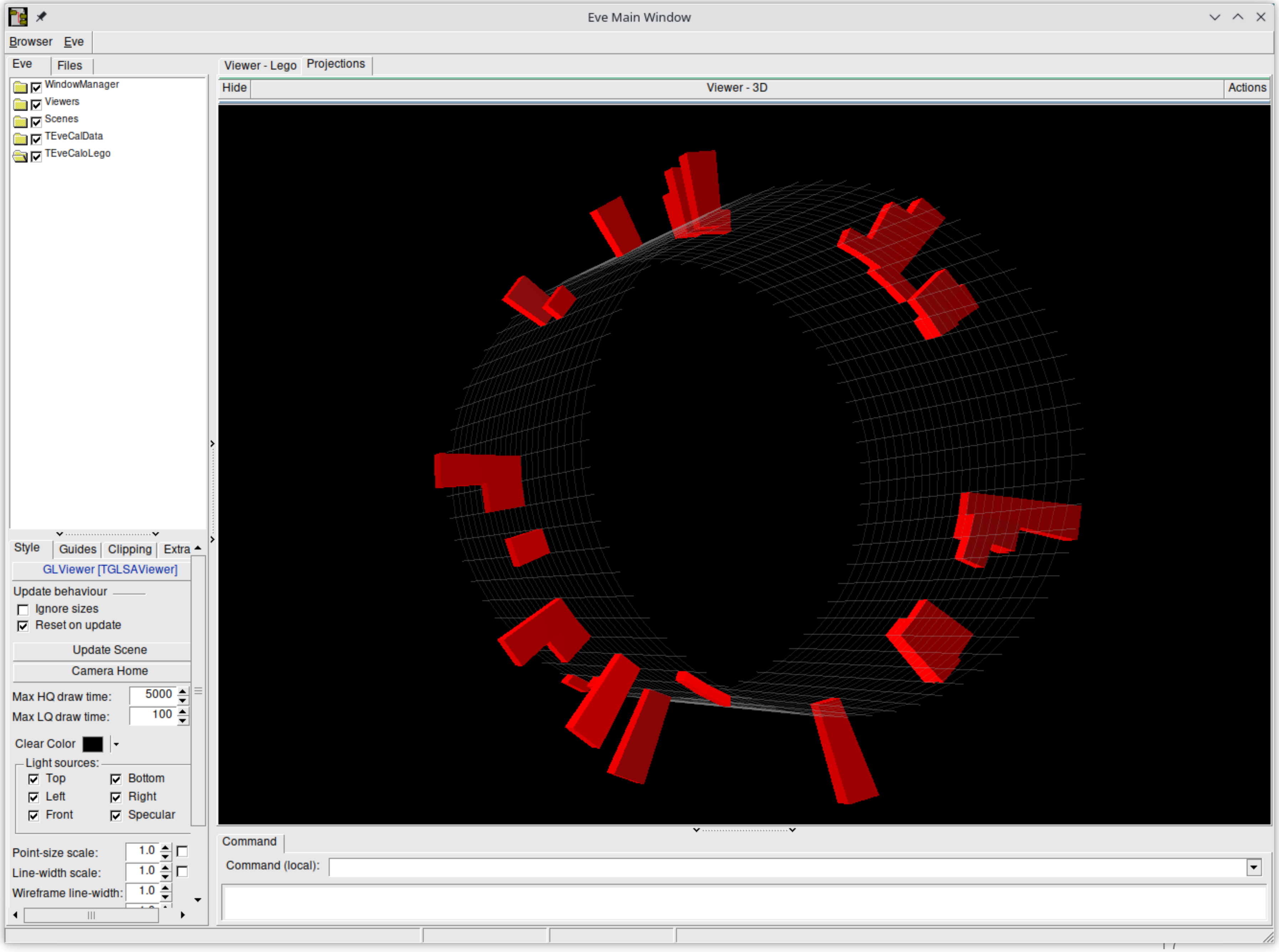
Hitformat

```
$ dlist oHCal-00000100-0000.evt
Packet  8001  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8002  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8003  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8004  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8005  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8006  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8007  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8008  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8009  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8010  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8011  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8012  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8013  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8014  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8015  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8016  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8017  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8018  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8019  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8020  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8021  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8022  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8023  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8024  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8025  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8026  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8027  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8028  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8029  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8030  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8031  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
Packet  8032  1000 -1 (sPHENIX Packet)  92 (IDDIGITIZERV1)
```

oHCal Data

As a quick exercise, I took our oHCal events and made an Event Display

- Final packet access API
- What we are doing here (with cosmics) will look (structure-wise) exactly like our data next year
- People can work with the data for calibration procedures, verify channel mappings, interface F4A with the real data structure, and on on and on
- Tons of test beam data exist that have already been analyzed but can answer additional questions
- (and yes, one can make Event Displays…)



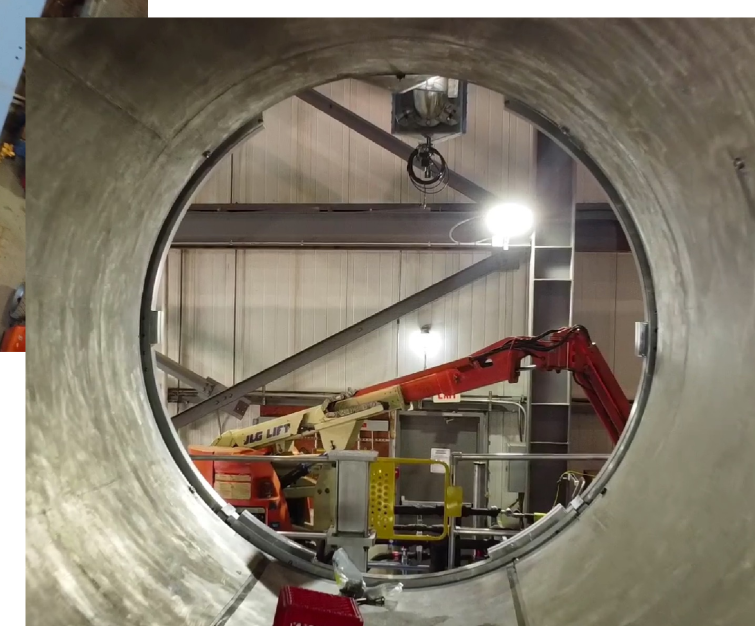
Fly through the EPIC Detector!

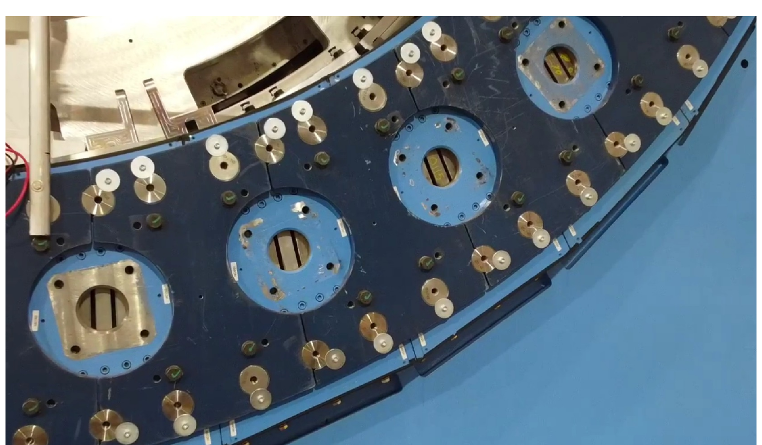
We had the opportunity to fly a drone through the current installation with the outer Hcal and the magnet

Those 2 will be part of EPIC

Take a flight through the detector! Go to

https://www.phenix.bnl.gov/~purschke/Drone/cut01.mp4





Summary

Solid proposal that revolves around the concept of a Data Aggregation Module (DAM) and the existing and rock-solid RCDAQ system

Today: DAM==FELIX (that cannot be built any longer)

Several projects to bring the “next FELIX” into the next decade under way or on the horizon

A modest amount of new ASICs for the front-ends (didn’t have time to talk about that)

Envisioned data rates/volumes manageable even by today’s standards

(off the cuff: that usually leads to great new ideas what to do with that bandwidth!)

Lots of support and existing test beam data available for R&D-level DAQs (old eRDxes, 1,6,23,…) and new eRD108, eRD110, …

Backup

Data Reduction/Compression

Every detector obviously wants to minimize the data volume without losing physics information

Lossy: zero-suppression (threshold), clustering w/ threshold, etc

Zero-suppression is a must to avoid clogging up the front-end pipes.

However:

We can apply loss-less compression as a catch-all to offset compromises in threshold settings

Also, the early data are not as ”dense-packed” – development/learning curve requiring actual data

Set thresholds as low as the front-end bandwidth allows, let late(r)-stage compression do the rest

For 2 decades we have always applied late-stage, distributed compression to our raw data (PHENIX/sPHENIX)

Distributed: happens at the EBDC stage

Rock solid, and even saves significant time at the reconstruction stage

(reading less data vastly over-compensates the small penalty for the decompression step)

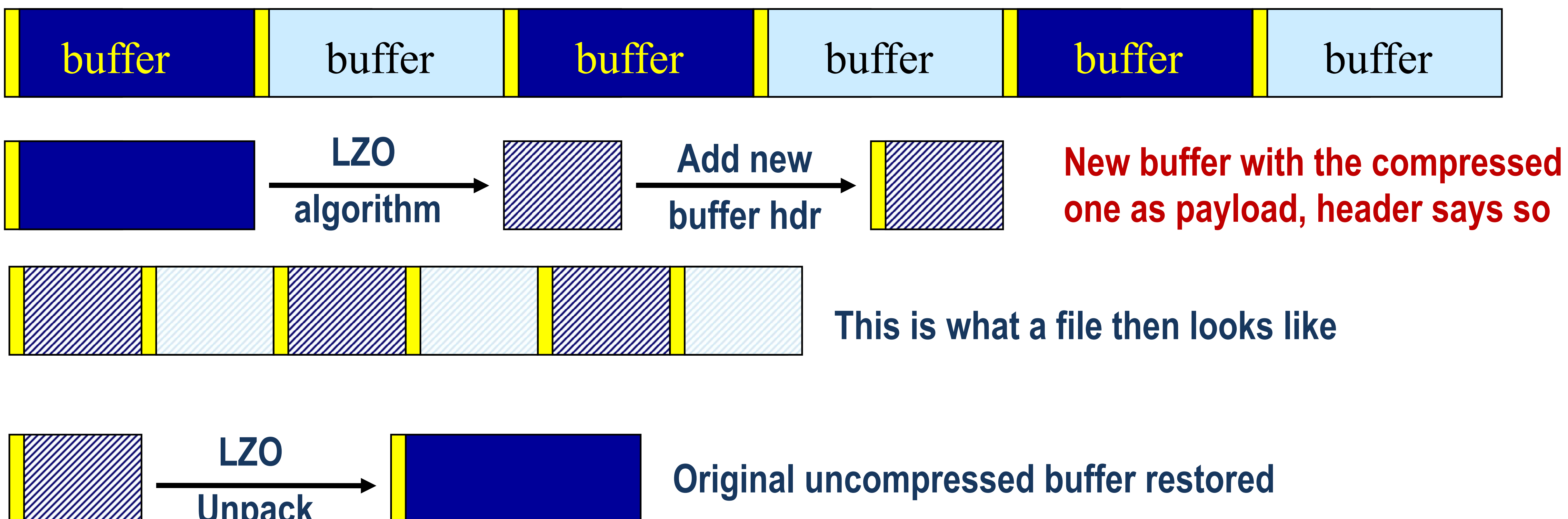
Conceptually similar to compressed root files, but different/much faster compression algorithm

Data Compression

After all data *reduction* techniques (zero-suppression, bit-packing, etc) are applied, you typically find that your raw data are still compressible to a significant amount

Our compressed raw data format supports a late-stage data compression:

This is what a raw data file would normally look like (a buffer typically holds 10-500 events, e.g. 64MB)



All this is handled completely in the I/O layer, the higher-level routines just receive a buffer as before.

Compressed data

The current ECCE test bench/test beam/etc data that we take are super-compressible

no or super-low occupancy, not zero-suppressed, etc

compression **down to** ~5% of the original, not typical for the “real” running

```
$ prdfcheck /data/phnxrc/1008/junk/hcal_lzo_00000100_0000.evt | more
buffer at record    0 length =  201799  25 marker = ffffbbfe  LZO Marker  Or.length: 4194208  4.81137%
buffer at record   25 length =  201304  25 marker = ffffbbfe  LZO Marker  Or.length: 4194264  4.79951%
buffer at record   50 length =  201424  25 marker = ffffbbfe  LZO Marker  Or.length: 4194264  4.80237%
```

Expect a 40% value (compression by 60%) in the early going, going up to low 70%s

(One can think of this as a metric for the actual “information content per bit”)

Late-generation PHENIX raw data (2016, last run):

```
$ prdfcheck  EVENTDATA_P00-0000443135-0001.PRDFF  | more
buffer at record    0 length = 3285885  402 marker = ffffbbfe  LZO Marker  Or.length: 4357160  75.4135%
buffer at record  422 length = 3064576  375 marker = ffffbbfe  LZO Marker  Or.length: 4349976  70.4504%
buffer at record  797 length = 3204863  392 marker = ffffbbfe  LZO Marker  Or.length: 4250952  75.3917%
```

Compression speed

"dpipe" is a swiss-army-knife utility to work with raw data. Take a file, uncompress, re-compress, manipulate, etc etc. The file here contains 77524 events.

```
$ time dpipe -sf -df -l EVENTDATA_P00-0000443135-0001.PRDFF EVENTDATA_P00_LZO-0000443135-0001.PRDFF

real    0m2.866s
user    0m2.354s
sys     0m0.507s
```

Asks for LZO-compression

```
$ bc -lq
2.866 / 77524 * 10^6
36.96919663588050000000

77524 / 2.866
27049.54640614096301465457
```

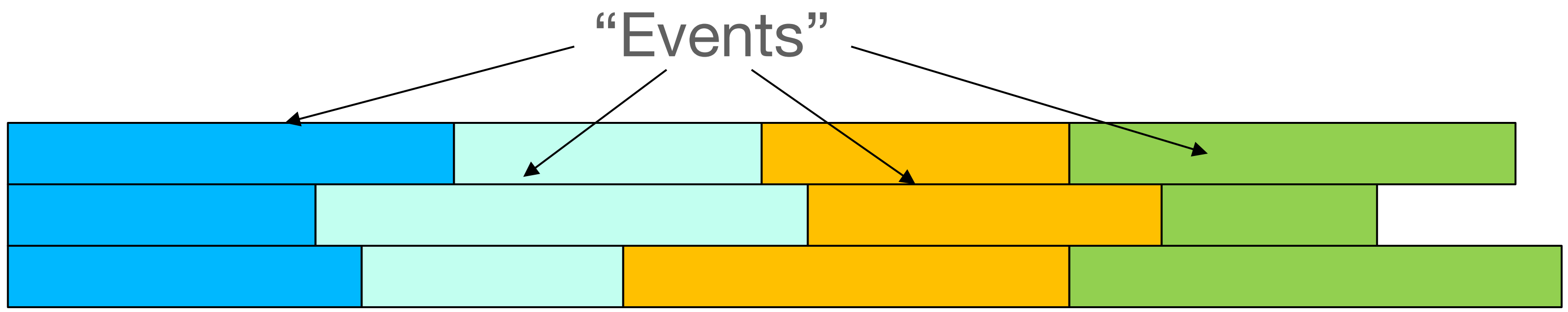
So 37μs per event and 3.5GBytes/s compression rate per thread

BTW – I keep a gzip-compression format around as a benchmark – this shows just how much faster LZO is compared to gzip, for only a 10% additional improvement

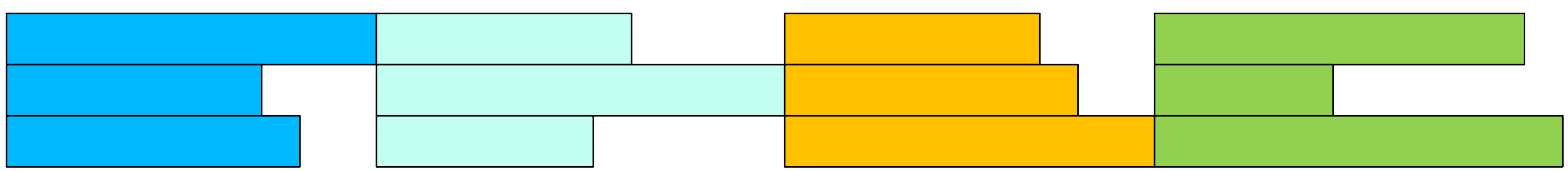
```
$ time dpipe -sf -df -z EVENTDATA_P00-0000443135-0001.PRDFF EVENTDATA_P00_gz_-0000443135-0001.PRDFF

real    6m58.935s
user    6m55.183s
sys     0m3.659s

$ ls -l   /mnt/ramdisk/*
-rwxr--r-- 1 phnxrc phnxrc 10739654656 May  9 08:15 /mnt/ramdisk/EVENTDATA_P00_LZO-0000443135-0001.PRDFF
-rwxr--r-- 1 phnxrc phnxrc  9161973760 May  9 08:22 /mnt/ramdisk/EVENTDATA_P00_gz_-0000443135-0001.PRDFF
```

Asks for gzip-compression

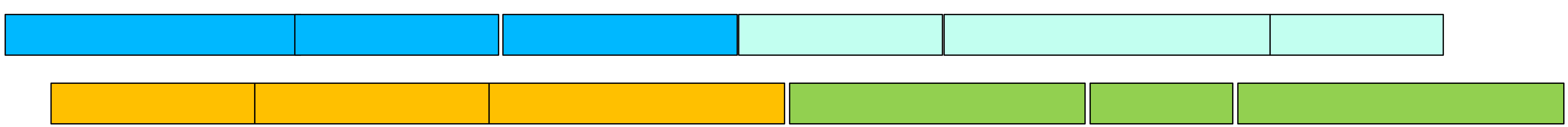
How do you deal with that?



You could throttle your event rate with busies to not let that happen:



Or, if you insist on "event boundaries" in your data, you could buffer those event fragments in DAQ memory, assemble them, then write out



Remember that you can often not "ask" a device to give you its data when it's convenient for you, you need to be ready to catch them as they come (e.g. network)

You have brought the event builder back, have to do it online, have to do it right the first time…

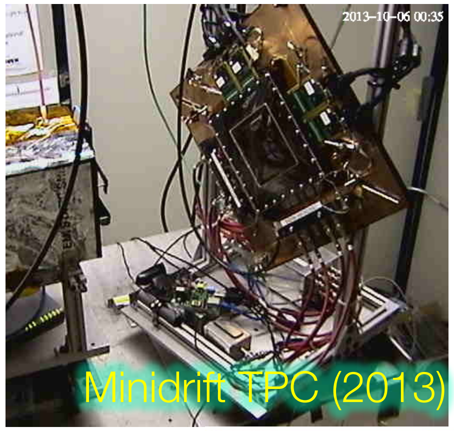
Why has the sPHENIX DAQ been important for the EIC and ECCE R&D?

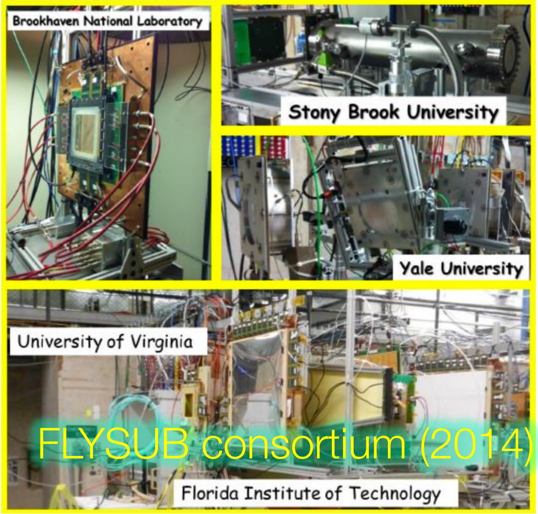
sPHENIX is one of the experiments paving the way for streaming readout in our community

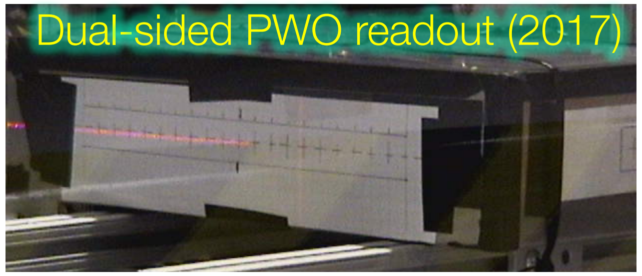
sPHENIX's RCDAQ system has been a pillar of EIC-themed data taking for R&D, test beams etc since 2013 – eRD1, eRD6, LDRDs, …

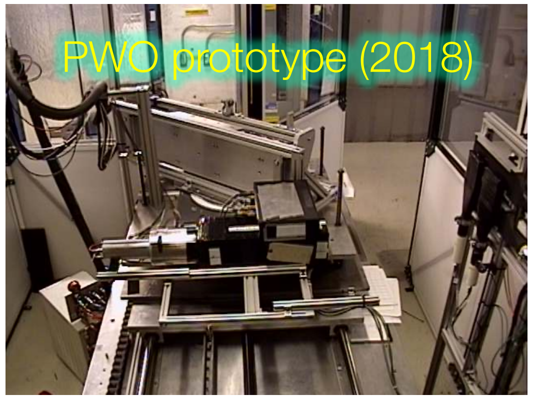
Estimated 25 active RCDAQ installations in the EIC orbit + ~30 elsewhere

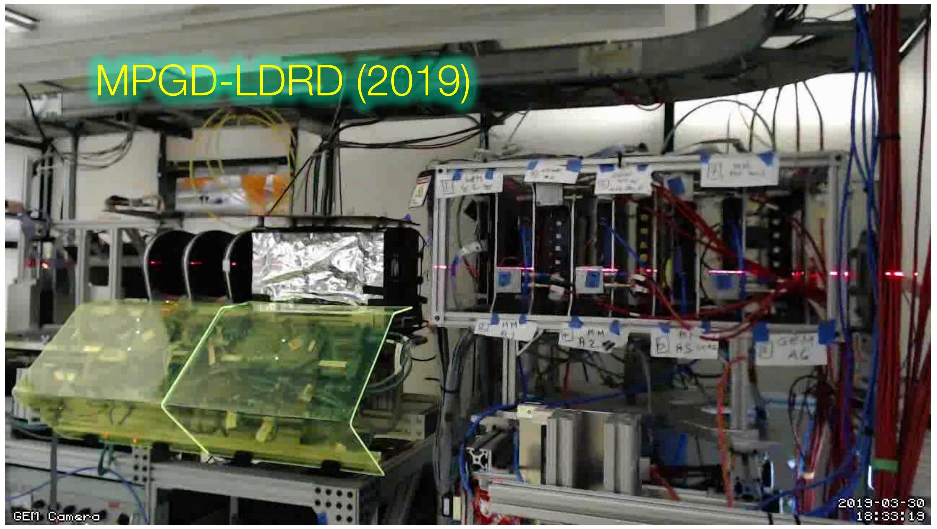
Usual entry by ease-of-use for standard devices (DRS, SRS, CAEN, …) and support for fully automated measurement campaigns



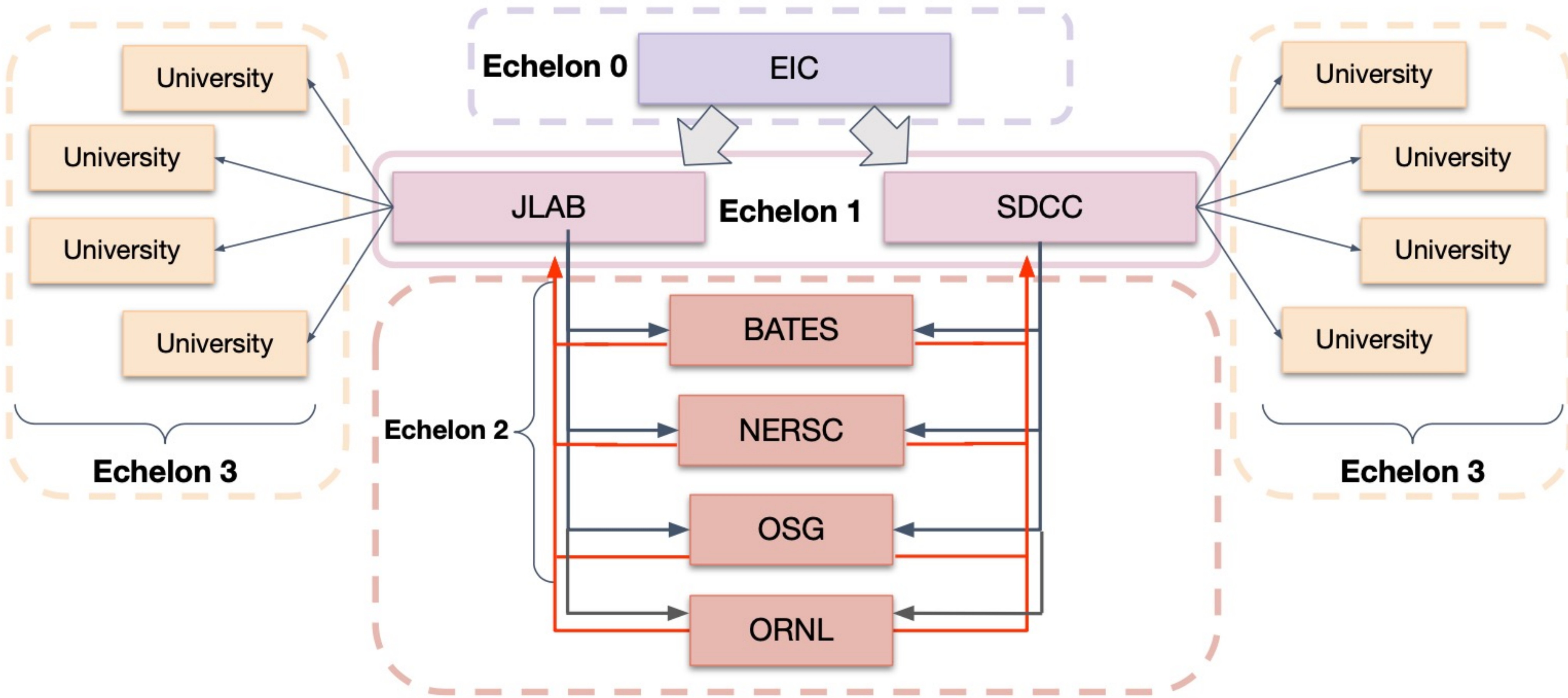








Connection to offline computing



Pretty standard GRID/distributed computing paradigm